An Introduction into Multigrid Method Fundamentals

An Honors Thesis (HONR 499)

by

Brian Pierson

Thesis Advisor

Dr. Ira Livshits

Ball State University
Muncie, Indiana

May 2019

Expected Date of Graduation

May 2019

Abstract

Numerical methods are a field of mathematics concerned with the creation of mathematical tools to solve applied problems. In this paper, we concentrate on multigrid methods, an approach that can be used for solving systems of linear equations that arise from discretizing ordinary differential equations, among many other things. Multigrid methods are an advanced topic not usually taught to undergraduates. Furthermore, self-learning is made challenging as most existing literature is written in a style typical to mathematics, which can be difficult to decipher for the inexperienced. However, the core concepts behind multigrid can be easily understood. In this paper, I provide introductions to these central ideas (*iterative methods and the smoothing property, coarse grids, residual correction, transfer operators,* and *recursion*), with the intention of bridging the knowledge gap that exists before more popular multigrid literature can be approached.

Acknowledgments

I would like to thank Dr. Ira Livshits for introducing me to multigrid, and for advising me in this project. Her knowledge and patience was critical to this paper, as the former gave me the foothold I needed to understand the topic on my own, while the latter was tested far too many times over this projects duration.

Contents

1 Process Analysis Statement

In the spring of 2018, I started studying multigrid methods as a potential topic to research for my senior mathematics capstone class at Ball State University. As someone who has always been more excited by the application of mathematics than by pure theory alone, I found the topic immediately interesting. However, I quickly found that I was unable to effectively learn from the existing literature. This was primarily caused by two reasons: first, I had little to no knowledge on the underlying principals multigrid methods operate off of. Second, as an undergraduate I was not yet comfortable with the traditional mathematics writing style, which can be difficult to decipher. Fortunately, I was able to learn the knowledge I needed to begin to understand the concept from my advisor, Dr. Ira Livshits. However, it was not lost on me that without her guidance I would not have been able to make any progress in the field. Thus, when the time came to choose my honors thesis, writing a paper that would have filled that gap for my former self seemed like a natural choice.

Multigrid as a concept is undoubtedly an advanced field of mathematics, and would not typically fall in the purview of an undergraduate. However, I personally found that the fundamental parts were straightforward to comprehend when separated and magnified. Furthermore, that comprehension allowed me to successfully return to multigrid literature I had found too challenging before. As such, what follows in this paper is an introduction to these same components of multigrid, with some context provided as to how they fit into multigrid as a whole.

Ideally, after finishing this paper the reader will have a broad understanding of what each essential part of multigrid does, as well as generally be familiar with their place in the overall structure. These ideas are presented in a more conversational tone than is typical for a mathematics paper, with the goal of reaching less experienced mathematicians like my former self. Numerically heavy sections are also backed up with graphics, where applicable. Lastly, I try to provide some context as to why multigrid exists in the first place, by presenting comparisons to some traditional methods that may be familiar to the reader. Overall, the goal of this paper is not to serve as a technical analysis of why multigrid works or a crash course to every minute detail. Instead, it's intended purpose is to be a primer or stepping stone for some future lost student like I once was. I hope you find it useful as well.

2 Introduction to Multigrid

Numerical analysis is a field of mathematics concerned with the creation of mathematical tools to solve applied problems. Modern science is an important motivator for the growth of this field, as it generates problems that are

often both large and need to be solved quickly. This encourages faster and more efficient numerical methods to be developed. In this paper, we concentrate on *multigrid*, a numerical method that can be used for solving systems of linear equations that arise from discretizing ordinary differential equations among many other things. Multigrid's unique approach is to get different types of information from the problem on different scales. This results in multigrid requiring comparatively fewer computations and less storage space to execute than most other solvers.

2.1 Example Problem

Before direct comparisons of numerical methods can be made, it's important to note multigrid methods require more tailoring to a specific application than other traditional solvers. As such, it will be beneficial to define a standard problem now before beginning explanation in depth.

For the purposes of this paper, I chose the second-order linear ODE boundary value problem:

$$u''(x) = f(x), \tag{1}$$

with the Dirichlet boundary conditions:

$$u(a) = u_a, \tag{2}$$
$$u(b) = u_b. \tag{3}$$

This problem is approximated by the finite difference scheme:

$$\frac{u_{i-1}^h - 2u_i^h + u_{i+1}^h}{h^2} = f_i \text{ for } i = 1, \ldots, n, \tag{4}$$

where $u_i^h \approx u(x_i)$, $f_i^h = f(x_i)$, $u_0^h = u_a$ and $u_{n+1}^h = u_b$.

3 Review of Solvers

In order to better understand the advantages of multigrid, let's briefly review the traditional families of numerical solvers.

3.1 Direct Methods

Direct methods of solving systems of linear equations are characterized by the ability to find an exact solution in a finite number of steps. In theory, no approximation is required when implementing these methods, meaning the solution obtained is exact. LU Factorization and/or Gaussian elimination are two common examples of these methods.

This exact solution comes at the cost of efficiency: for example, Gaussian Elimination has a computational complexity of $O(n^3)$ in the worst case scenario. In other words, the number of operations needed to compute the solution is proportional to the grid size cubed. This inefficiency in computational time means that systems can often be too large to be solved directly in a time frame that's reasonable for the problem.

Numerical solvers are often executed with the help of a computer, which brings in another disadvantage for direct methods. Computers inherently will introduce rounding errors during the computation process due to memory constraints. These errors will compound over each successive operation, rendering the theoretically exact solution inaccurate. The rounding errors and high number of calculations play into each other, limiting the cases where direct methods can be applied effectively. Thus, alternative methods are required.

3.2 Iterative Methods

In order to combat some of these weaknesses of direct methods, let's look at the concept of *iterative methods*. These techniques focus not on directly finding the exact solution, but instead on using known information to repeatedly estimate a better solution. Typically, this means the cost of computation can be greatly reduced in comparison to direct methods, while still being able to achieve any desired accuracy. For an example iterative method, let's examine *Gauss-Seidel relaxation*.

The basic operating principle of Gauss-Seidel relaxation is to break the matrix A into it's basic components of a strictly lower triangular matrix L, a diagonal matrix D, and a strictly upper triangular matrix U, such that:

$$Av^n = (L + D + U)v^n = b. \tag{5}$$

We do this because each of those components are easy to invert on their own, as opposed to inverting the whole. It is trivial to show

$$(L + D)v = b - Uv \tag{6}$$

to be true. However, if we use an approximation for v, say u, on the right side, we will find we obtain an improved approximation u' on the left. Rewriting the equation to solve for this new approximation, we have:

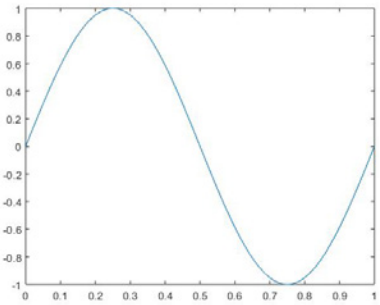
$$u' = (L + D)^{-1}(b - Uu) \tag{7}$$

We can also take advantage of the triangular form of $(L + D)$ to make further improvements to the method. Namely, we can use previously calculated elements in u' when determining later elements in u', improving the approximation overall. This process is called *forward substitution*.

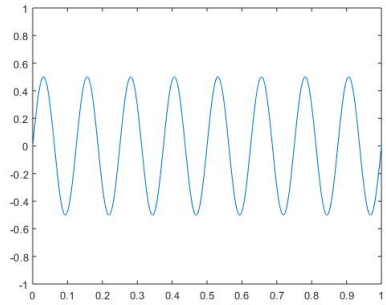
3.3 Error

Looking forward, it's important to address a feature shared by many relaxation schemes called the *smoothing property*. However, let's first standardize the way in which we talk about error in numerical systems. Error can be simply defined as the distance from the current approximation to the exact solution. (Remember that our solution is the exact v so that $Av = b$ for a given A and b.) This distance is critical to iterative methods, as it is only by reducing the error every iteration that the error can eventually be reduced to zero. Here, it's useful to talk about error in terms of compositions of waves, or oscillations. This is because error can often best be represented as the sum of many separate oscillations of varying period and magnitude. Let's look at an example:

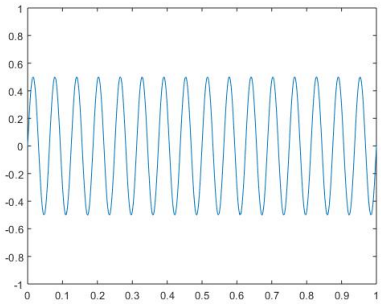
Say we have a current approximation u to the exact solution v. Then, we have error $e = v - u$. Let's say our error is created from the combination of three oscillatory functions, as they appear in Figure 1.



(a) Low Oscillatory Function



(b) Mid Oscillatory Function



(c) Highly Oscillatory Function

Figure 1: Error Components

We define our error to be the sum of these three components, and hence our error e currently appears in as Figure 2.

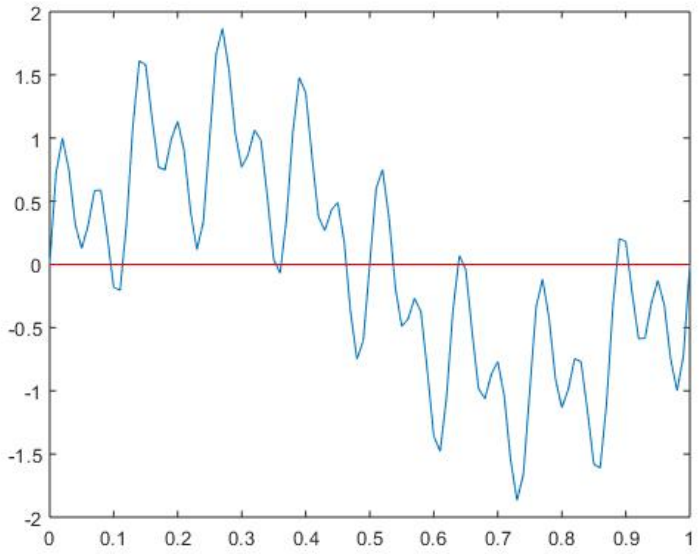
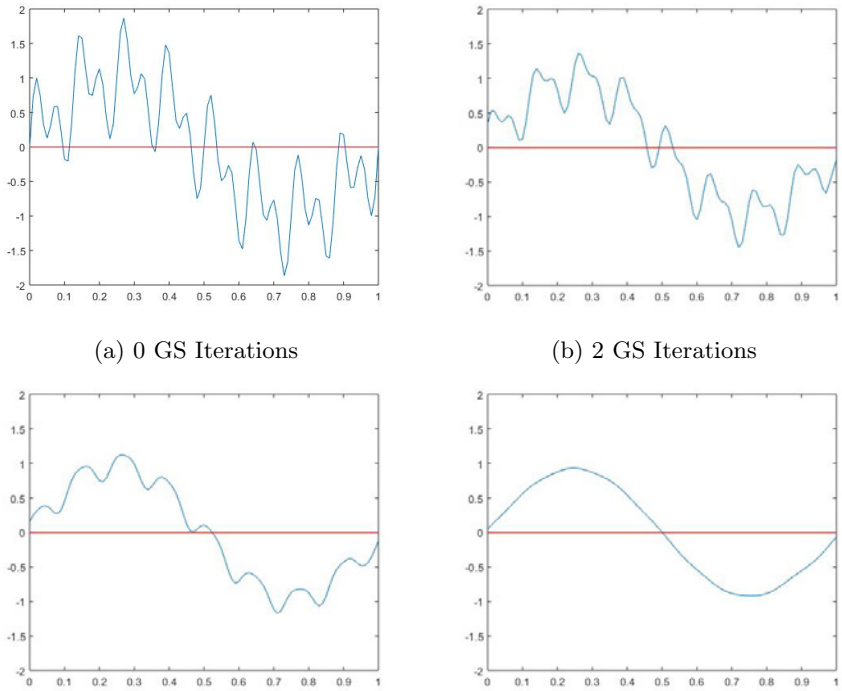


Figure 2: Plotted Error

As we can see, the error appears to be chaotic and jagged, despite only being comprised of three functions. These functions are called the oscillatory components of the error. In general, any error can be described using a combination of any number of these components. Note that this example constituted a highly simplified case, and it is not difficult to imagine that error consisting of many, many components may be nearly indistinguishable from random noise.

3.4 Smoothing Property

Now that we have a standardized way to describe error, let's examine how it is effected by Gauss-Seidel Relaxation. For this example, we derive our matrix A from the second-order linear ODE boundary value problem we detailed earlier, and set our desired solution to be the zero vector. Thus, our current approximation u also serves as our error, as $e = u - \vec{0} = u$. For comparisons sake, let's take our initial guess to be the error we described in the last section. Here's how the error appears after 0, 2, 5, and 20 iterations of GS relaxation: (Figure 3 next page).



(a) 0 GS Iterations

(b) 2 GS Iterations

(c) 5 GS Iterations

(d) 20 GS Iterations

Figure 3: Demonstration of Gauss-Seidel Smoothing Error

Notice how the error components that are high frequency in nature are quickly reduced, leaving only the middle and lower frequency components after only 5 iterations. This is the *smoothing property* in action. In essence, the smoothing property says that iterative methods will eliminate high frequency (usually described as *oscillatory*) error components effectively, but will work slowly on low frequency components (usually described as *smooth*). This property is one of the major downsides to iterative methods, as it means approximating solutions with high accuracy can take an unacceptable number of iterations, particularly if the original approximation was poor. We can see this in our example, as even after 100 iterations we are a long way from the solution: (Figure 4 next page)

4 Multigrid

As we have shown, the *smoothing property* shared by many iterative methods can be a major limitation. Although the oscillatory components of error are dampened quickly and efficiently, the smooth components are not eliminated.

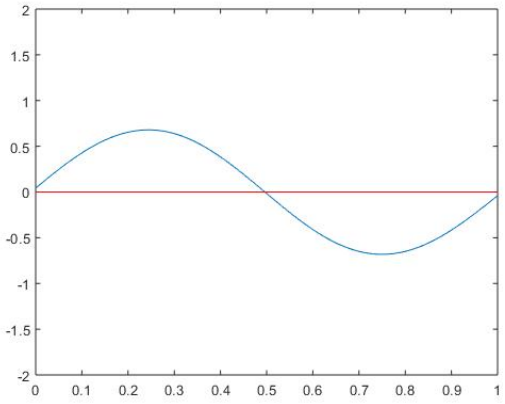


Figure 4: Error After 100 GS Iterations

Typically, the smoothing property is an unavoidable feature of iterative methods. However, if we could find a way to treat smooth error as oscillatory, then we could reduce all error components with the same efficiency that oscillatory error is reduced with. In effect, we could re-contextualize the smoothing property from a weakness into a strength.

4.1 Coarse Grids

Enter the idea of *coarse grids*. Up to this point, we have been assuming our problem is discretized on a grid of some set size n. However, it is not a requirement that we operate on grid size n specifically, as the problem can be approximated on any desired grid size. Let's examine what happens when we reduce to a grid size of $n/2$. Take for example this smooth function, sampled 8 times over it's period. (Figure 5)

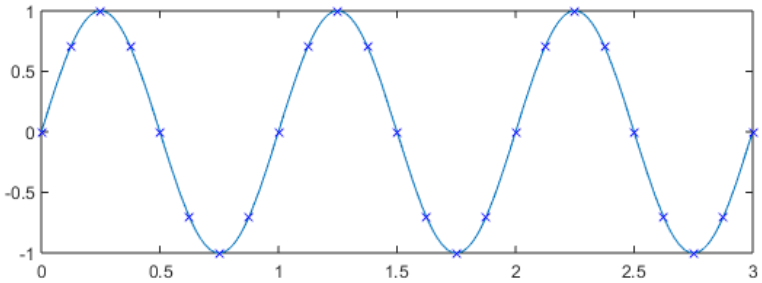


Figure 5: A Smooth Function

We can create a linear interpolation of this function, using points regularly sampled over its domain. (Figure 6)

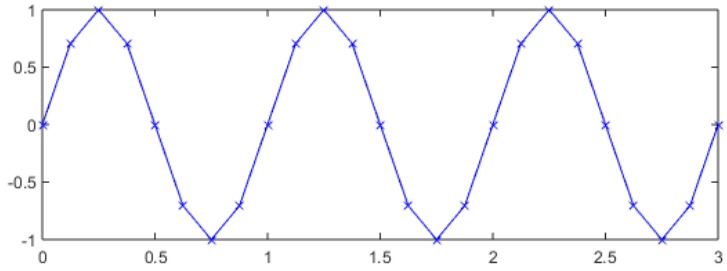


Figure 6: Fine Grid Linear Interpolation

We see that although some information is lost, the linear interpolation is still a decent approximation of the original function. Now, if we halve the grid size, we reduce the number of sampled points by half, resulting in this linear interpolation on the coarser grid. (Figure 7)

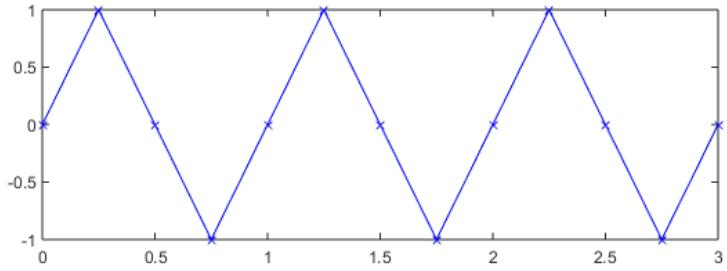


Figure 7: Coarse Grid Linear Interpolation

In this example, the function on the fine grid repeats every eight points, where as on the coarse, the function repeats every four. The result is the function sampled on the coarse grid is perceived to be of relatively higher frequency than when it was sampled on the fine grid. This means the relaxation performed on the coarse grid will be more effective than relaxation performed upon the fine grid.

Coarse grids also offer computational advantages on top of changing the perceived frequency of functions. Recall the cost of iterative methods like GS relaxation is directly proportional to the number of points being operated on. Thus, reducing the grid size also decreases the computational complexity of the operation.

4.2 Residual Correction

Thus far, we have established that iterative methods like Gauss-Seidel Relaxation feature the smoothing property. This indicates that oscillatory error will be damped effectively, while smooth error will persist. We have also identified coarse grids to be advantageous by providing a way of treating smooth error components as oscillatory, and by being computationally cheaper to operate on. Thus, it seems advantageous to switch between grid sizes when it would provide a computational advantage. However, we have yet to establish what form the original fine grid problem will take on the coarse grid.

Residual correction provides us with a straight forward method of transferring problem information in-between grid sizes. It is an iterative way to reduce error in a numerical system, although it is not a solver like Gauss-Seidel Relaxation. Let's go over it's derivation: say we have u, an approximation for v in $Av = b$. We are already familiar with the error between the approximate and exact solutions, $e = v - u$. Less familiar is the *residual* r, which is the difference between the intended result b and our current result Au:

$$r = b - Au. \tag{8}$$

If we knew the error e, then correcting u would be trivial, as $u + e = v$. However, unlike e the residual r is a known quantity, and it can be shown e is an exact solution to $Ae = r$:

$$\begin{aligned} r &= b - Au \\ r &= Av - Au \\ r &= A(v - u) \\ r &= Ae \end{aligned} \tag{9}$$

Since v is unknown, we solve for e in $r = Ae$ using any iterative method of our choosing, resulting in an approximation of the actual error $\hat{e} \approx e$. We use this value to improve our original approximation u, resulting in a better approximation u'.

$$u' = u + \hat{e} \tag{10}$$

Let's look at how residual correction can be used to make transferring our problem between grid sizes easier. Transferring problem information from a fine grid to a coarse grid is fairly easy to do, so we create $r^H = A^H e^H$ from $r^h = A^h e^h$[1]. We can then iteratively solve for e^H, using a random initial guess. This is the main advantage of residual correction, as e^H contains only new

[1]Remark on notation: It's common practice to indicate elements that exist on the coarse grid with a superscript H, and elements that exist on the fine grid with a superscript h. This is in reference to the distance between grid points, and will be the standard notation we use from here on out.

information determined from relaxation. As such, it can be easily transferred back up to the fine grid to be directly added to u^h:

$$e^H + u_0^h = u_1^h \approx v \tag{11}$$

4.3 Transfer Operators

However, e^H and u^h still cannot be added directly due to differing grid sizes. We need to define exact methods for transferring problem information from one grid size to another. The way in which grid sizes can change when going from fine to coarse varies between multigrid implementations, so there is no set standard for how this transfer will work. For a specific example, let's explore a simple choice that works for our example problem, full weighting and linear interpolation.

4.3.1 Prolongation

We first examine *prolongation*, or the mapping from the coarse grid to the fine grid. We do this via *linear interpolation*, the method for which we denote I_H^h:

$$I_H^h : grid\ n/2 \longrightarrow grid\ n \tag{12}$$

If we have v^h, v^H as vectors on grid n and $n/2$, respectively, then we define I_H^h as the function such that:

$$I_H^h v^H = v^h \tag{13}$$

For proper linear interpolation, we want the points on both the coarse and fine grid to remain unchanged between the two, while the points that exist on the fine grid but not on the coarse grid to be the average of their neighbors. (Figure 8)

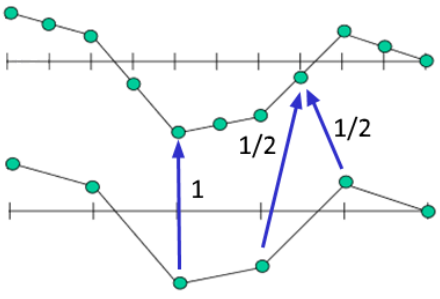


Figure 8: Prolongation Visualized

Thus we say for $i \subset \mathbb{R}$:

$$0 \leq i \leq \frac{N}{2} - 1 \begin{cases} v_{2i}^h = v_i^H \\ v_{2i+1}^h = \frac{1}{2}(v_{i-1}^H + v_{i+1}^H) \end{cases} \tag{14}$$

From this we can build the desired linear interpolation operator.[2] For example, if $n = 7$:

$$\begin{bmatrix} \frac{1}{2} & & \\ 1 & & \\ \frac{1}{2} & \frac{1}{2} & \\ & 1 & \\ & \frac{1}{2} & \frac{1}{2} \\ & & 1 \\ & & \frac{1}{2} \end{bmatrix} \begin{bmatrix} v_1^H \\ v_2^H \\ v_3^H \end{bmatrix} = \begin{bmatrix} v_1^h \\ v_2^h \\ v_3^h \\ v_4^h \\ v_5^h \\ v_6^h \\ v_7^h \end{bmatrix} \tag{15}$$

4.3.2 Restriction

Restriction serves an inverse role to prolongation, namely the mapping from the fine grid to the coarse grid. Again, the exact details depend on the multigrid implementation, so let's look at a method called *full weighting*. We denote:

$$I_h^H : grid\ n \longrightarrow grid\ n/2 \tag{16}$$

For full weighting, we wish for every point on the coarse grid to be the weighted average of the three nearest points on the fine grid, with twice as much weight placed on the center point. (Figure 9)

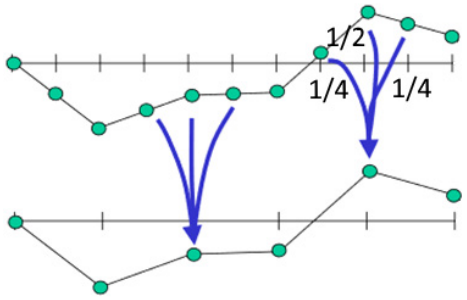


Figure 9: Restriction Visualized

[2]Note that the outer two most points on the fine grid, v_1^h and v_7^h, seem to only get half of a value from the coarse gird, not an average. This is a quirk of our example problem, in reality the rest of the average would come from the predefined boundary values.

Thus we say for $i \subset \mathbb{R}$:

$$v_i^H = \frac{1}{4}v_{2i-1}^h + \frac{1}{2}v_{2i-1}^h + \frac{1}{4}v_{2i}^h \tag{17}$$

We can then build an example operator, again with $n = 7$:

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & & & \\ & & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & \\ & & & & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix} \begin{bmatrix} v_1^h \\ v_2^h \\ v_3^h \\ v_4^h \\ v_5^h \\ v_6^h \\ v_7^h \end{bmatrix} = \begin{bmatrix} v_1^H \\ v_2^H \\ v_3^H \end{bmatrix} \tag{18}$$

In general, these two functions are called the transfer operators.

We now have nearly all the components we need to understand a basic multigrid implementation. Let's look a simple scheme: (Figure 10)

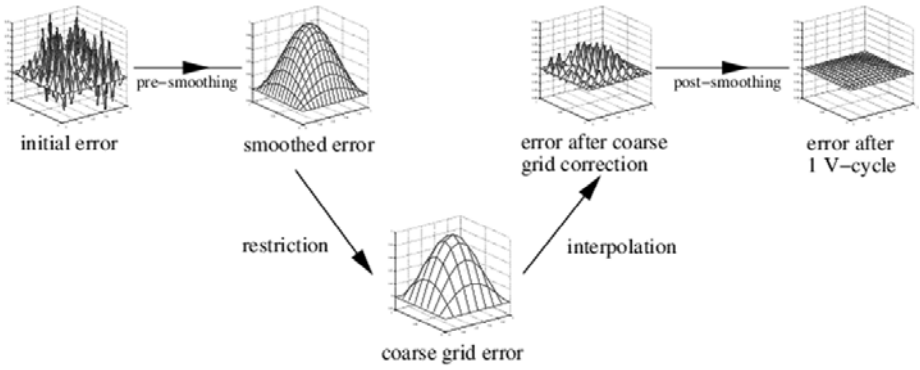


Figure 10: A Basic Multigrid Cycle

4.4 Solving on the Coarse Grid

The only thing left to be determined is how to solve the the residual equation $A^H e^H = r^H$ on the coarse grid. We've discussed several choices over the course of this paper, from direct methods to iterative methods, but in general the best choice is to simply use multigrid again. By treating the coarse grid problem as a new fine grid start, we can descend down to an even coarser grid. This option is bolstered by the fact that coarse grid correction only corrects some, but not all smooth error[3]. So, by going another layer deeper we can correct

[3]In our example problem we reduce the grid size by a factor of ≈ 2, so roughly half of the smooth error on the fine grid is treated as oscillatory on the coarse grid.

even more smooth error. This process of reapplying multigrid to itself is known as *recursion*.

Of course, we are not allowed to continue with this cycle forever, as at some point the grid size would decrease to a value less than 1. Part of designing a multigrid implementation is choosing the criterion at which to break the recursion loop. It's typical for the decision point to be based on reaching some predefined minimum grid size. The advantage of being able to descend from arbitrarily large grid sizes to a grid size of your choice is in the ability to choose a grid size that best suits the situation. For example, it's common to choose some small grid size at which solving for $A^H e^H = r^H$ using a direct method is no longer affected by the downsides present at large grid sizes. Thus you can quickly get an exact answer for the error at the coarsest grid, which can then be transferred all the way back to the finest grid.

5 Multigrid Methods

5.1 A Multigrid Scheme

We've now covered all the major ideas behind multigrid, and now should be able to understand a basic multigrid scheme:

1. Perform relaxation of $A^h v^h = b$ on grid h to reach approximation v_0^h (Arbitrary initial guess)
2. Find residual $r^h = b - A^h v_0^h$
3. Calculate $r^H = I_h^H r^h$
4. If grid size H is small enough to allow direct methods, proceed to Line 5. Otherwise, repeat from Line 1 with $b = r^H$
5. Directly solve $e^H = (A^H)^{-1} r^H$ on coarsest grid
6. Apply correction $v_1^h = v_0^h + I_H^h e^H$.
7. Perform further relaxation of $A^h v_1^h = b$ to reach approximation v_{final}^h
8. If the grid h is the original grid, then $v_{final}^h \approx u^h$.
   Otherwise, repeat from Line 6 with $e^H = v_{final}^h$

I find a visual[4] to be more effective than an instruction list alone: (see Figure 11)

[4] Apologies for the unfamiliar notation, the internet does a better job creating visuals than I could ever hope, so this is sourced with appreciation from [4]

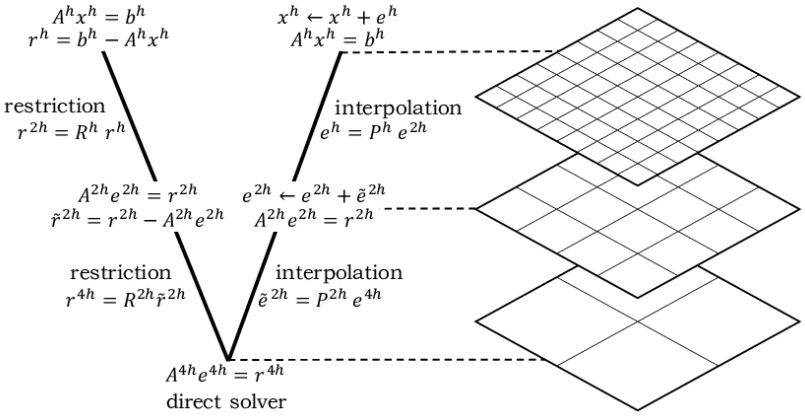


Figure 11: The Multigrid Cycle

5.2 Suggested Further Inquiry

With the main ideas behind multigrid now introduced, the knowledge gap to mainstream multigrid texts should be more manageable. Should you have found your way to this paper from one of those documents, hopefully you can now progress further than you were able to before. If this paper was your first introduction to multigrid, then Prof. Achi Brandt's *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics*[1] is a good resource to turn to next. I also found the slides from *A Multigrid Presentation*[2] as presented by Dr. Van Emden Henson to be useful as an intermediate step. In general, the internet is a useful resource for multigrid. Once I understood the methods well enough to properly formulate my questions, I found I was able to get a useful result within the first page.

In addition to the above, I'd personally like to recommend implementing a multigrid solver yourself in code. Although this paper alone does not give all the knowledge required to finish this task, not many gaps are left to be filled. Personally, I found that I gained a far deeper understanding of multigrid after developing a working algorithm and seeing it run, as opposed to simply reading about it. Several code examples exist online, and with those as a guide I believe this would be fruitful task for anyone who wants to understand multigrid better.

6 Closing Thoughts

6.1 Concept Review

As some closing thoughts, here's a review of the major concepts covered in this paper:

- *Iterative methods* like *Gauss-Seidel Relaxation*: Solve v in $Av = b$ by using known information to repeatedly generate better approximations, $u \approx v$. These methods display the *smoothing property*, which represents their ability to reduce oscillatory error effectively, but are ineffective on smooth error.
- *Coarse Grids*: Changing the grid size from fine to coarse allows some smooth error to be treated as oscillatory, re-enabling effective reduction of error.
- *Residual Correction*: An iterative concept for reducing error, but needs to be paired with an actual solver produce error estimate e. It's results can transfer easily between grid sizes, without confusion as to what information is important.
- *Transfer Operators*: Mappings between the fine and coarse grid. Exact definition will change depending on problem context and grid structures.
- *Recursion*: By recursively applying multigrid methods to solve for the error on the coarse grid, the grid size can be reduced to any chosen level. This means direct methods could be used to solve for the error at the coarsest grid.

6.2 Motivation for Multigrid

As a rule of thumb, direct methods are situationally better than iterative methods, generally when the problem size is small or simple in complexity. However, multigrid will always outperform comparable iterative solvers in situations where such methods are desirable. As an example, let's directly observe how multigrid performs in comparison to Gauss-Seidel relaxation when solving our reference problem.

We do this by starting with the same random guess for each method, then comparing the norm of the error after each cycle. In order to keep the comparison fair, we would like the multigrid cycle and Gauss-Seidel cycle to be about equal in computational requirements. To achieve this, we must perform multiple Gauss-Seidel relaxation passes per one multigrid cycle. In particular, this multigrid implementation performs four GS relaxation passes per level, meaning our GS cycle should consist of ten GS relaxation passes performed on the finest grid. Figure 12 demonstrates the results on the next page.

# Of Cycles Completed	Gauss-Seidel Relaxation L2 Norm of Error	Gauss-Seidel Relaxation % Error Reduction	Multigrid Method L2 Norm of Error	Multigrid Method % Error Reduction	% Difference (MG-GS)/GS
0	22383	-	22383	-	0
1	457.13	97.9601	2455.7	89.0287	437.1994
2	168.63	63.1112	298.96	87.8259	77.2875
3	111.1	34.1161	38.177	87.2301	-65.6373
4	87.971	20.8182	5.3265	86.0479	-93.9452
5	74.233	15.6165	.71526	86.5717	-99.0365
6	64.644	12.9174	.097553	86.3612	-99.8491
7	57.381	11.2354	.013177	86.4925	-99.9770
8	51.61	10.0573	.001708	87.0388	-99.9967
9	46.891	9.1436	.000251	85.3042	-99.9995

Figure 12: Gauss-Seidel and Multigrid Performance Comparison

Although GS relaxation holds an advantage early, the multigrid method quickly overcomes it's head start in error reduction. This is due to the fact that MG is able to produce consistent levels of error reduction cycle over cycle, while GS relaxation slows due to diminishing returns. Note that my personal code used for this example has several inefficiencies present, so in an ideal situation multigrid should produce an even better result of $\approx 90\%$ reduction every cycle.

As a trade off for this efficiency, multigrid methods are more challenging to implement than other comparable solvers. Decisions must be made on appropriate grid differences, iteration counts and problem parameters. Furthermore, the complexity and uniqueness of each multigrid instance makes troubleshooting any problems that arise difficult. This results in multigrid methods being far from a "one-size-fits-all" idea, and implementation can often be non-trivial.

6.3 Conclusion

Overall, multigrid methods are worthwhile not because they're easy, but because they are effective. Their challenge lies in the extra development required to produce a working algorithm. That pain is balanced by their benefit, found in the ability to produce consistent reductions in error iteration after iteration. This advantage is a quality unique to multigrid when compared against the traditional iterative solvers.

As stated at the beginning, the goal of this paper was to provide a broad understanding of what each essential part of multigrid does, as well as generally help the reader be familiar with their place in the overall structure. I hope this paper has served it's purpose for you in that regard, and that you will be able to approach multigrid topics with a deeper understanding than you had before.

If not, then I hope this paper has inspired new questions about multigrid that had not been considered before, and that this work will provide a starting place to answer those questions through your own personal research.

References

[1] A. Brandt. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics*. Schloss Birlinghoven: GMD, 1984.

[2] W. L. Briggs. A multigrid tutorial. `https://www.math.ust.hk/~mawang/teaching/math532/mgtut.pdf`. Presented by Van Emden Henson.

[3] R. Falgout, Y. Ulrike, and L. Ruipeng. Multigrid and multilevel methods. *FASTMATH*. `https://fastmath-scidac.llnl.gov/research/multigrid-and-multilevel-methods.html`.

[4] H. Ibeid, L. Olson, and W. Gropp. FFT, FMM, and multigrid on the road to exascale: performance challenges and opportunities, 10 2018. `https://www.researchgate.net/publication/328599327_FFT_FMM_and_Multigrid_on_the_Road_to_Exascale_performance_challenges_and_opportunities`.

[5] V. Marra. On solvers: The v-cycle multigrid. *COMSOL Blog*, April 2016. `https://www.comsol.com/blogs/on-solvers-v-cycle-multigrid/`.

[6] M. M. Sussman. Multigrid solvers. `http://www.math.pitt.edu/~sussmanm/3040Summer14/multigrid.pdf`, June 2014.